CHALMERS
UNIVERSITY OF TECHNOLOGY

Code Constructions for Distributed Storage With Low Repair Bandwidth and Low Repair Complexity

Code Constructions for Distributed Storage With Low Repair Bandwidth and Low Repair Complexity

Siddhartha Kumar, *Student Member, IEEE*, Alexandre Graell i Amat, *Senior Member, IEEE*, Iryna Andriyanova, *Member, IEEE*, Fredrik Brännström, *Member, IEEE*, and Eirik Rosnes, *Senior Member, IEEE*

***Abstract*—We present the construction of a family of erasure correcting codes for distributed storage that achieve low repair bandwidth and complexity at the expense of a lower fault tolerance. The construction is based on two classes of codes, where the primary goal of the first class of codes is to provide fault tolerance, while the second class aims at reducing the repair bandwidth and repair complexity. The repair procedure is a two-step procedure where parts of the failed node are repaired in the first step using the first code. The downloaded symbols during the first step are cached in the memory and used to repair the remaining erased data symbols at minimal additional read cost during the second step. The first class of codes is based on MDS codes modified using piggybacks, while the second class is designed to reduce the number of additional symbols that need to be downloaded to repair the remaining erased symbols. We numerically show that the proposed codes achieve better repair bandwidth compared to MDS codes, codes constructed using piggybacks, and local reconstruction/Pyramid codes, while a better repair complexity is achieved when compared to MDS, Zigzag, Pyramid codes, and codes constructed using piggybacks.**

***Index Terms*—Codes for distributed storage, piggybacking, repair bandwidth, repair complexity.**

I. Introduction

In recent years, there has been a widespread adoption of distributed storage systems (DSSs) as a viable storage technology for Big Data. Distributed storage provides an inexpensive storage solution for storing large amounts of data. Formally, a DSS is a network of numerous inexpensive disks (or nodes) where data is stored in a distributed fashion. Storage nodes are prone to failures, and thus to losing the stored data. Reliability against node failures (commonly referred to as fault tolerance) is achieved by means of erasure correcting codes (ECCs). ECCs are a way of introducing structured redundancy, and for a DSS, it means addition of redundant nodes. In case of a node failure, these redundant nodes allow complete recovery of the data stored. Since ECCs have a limited fault tolerance, to maintain the initial state of reliability, when a node fails a new node needs to be added to the DSS network and populated with data. The problem of repairing a failed node is known as the repair problem.

Current DSSs like Google File System II and Quick File System use a family of Reed-Solomon (RS) ECCs [1]. Such codes come under a broader family of maximum distance separable (MDS) codes. MDS codes are optimal in terms of the fault tolerance/storage overhead tradeoff. However, the repair of a failed node requires the retrieval of large amounts of data from a large subset of nodes. Therefore, in the recent years, the design of ECCs that reduce the cost of repair has attracted significant attention. Pyramid codes [2] were one of the first code constructions that addressed this problem. In particular, Pyramid codes are a class of non-MDS codes that aim at reducing the number of nodes that need to be contacted to repair a single failed node, known as the repair locality. Other non-MDS codes that reduce the repair locality are local reconstruction codes (LRCs) [3] and locally repairable codes [4], [5]. Such codes achieve a low repair locality by ensuring that the parity symbols are a function of a small number of data symbols, which also entails a low repair complexity, defined as the number of elementary additions required to repair a failed node. Furthermore, for a fixed locality LRCs and Pyramid codes achieve the optimal fault tolerance.

Another important parameter related to the repair is the repair bandwidth, defined as the number of symbols downloaded to repair a single failed node. Dimakis *et al.* [6] derived an optimal repair bandwidth-storage per node tradeoff curve and defined two new classes of codes for DSSs known as minimum storage regenerating (MSR) codes and minimum bandwidth regenerating (MBR) codes that are at the two extremal points of the tradeoff curve. MSR codes are MDS codes with the best storage efficiency, i.e., they require a minimum storage of data per node (referred to as the sub-packetization level). On the other hand, MBR codes achieve the minimum repair bandwidth. Product-Matrix MBR (PM-MBR) codes and Fractional Repetition (FR) codes in [7] and [8], respectively, are examples of MBR codes. In particular, FR codes achieve low repair complexity at the cost of high storage overheads. Codes such as minimum disk input/output repairable (MDR) codes [9] and Zigzag codes [10] strictly fall under the class of MSR codes. These codes have a high sub-packetization level. Alternatively, the MSR codes presented in [11]–[18] achieve the minimum possible sub-packetization level.

Parts of this paper were presented at the IEEE Global Communications Conference (GLOBECOM), San Diego, CA, December 2015. This work was partially funded by the Research Council of Norway (grant 240985/F20), Simula@UiB, and the Swedish Research Council (grant #2016-04253).

S. Kumar was with the Department of Electrical Engineering, Chalmers University of Technology, SE-41296 Gothenburg, Sweden. He is now with Simula@UiB, N-5020 Bergen, Norway (e-mail: kumarsi@simula.no).

A. Graell i Amat and F. Brännström are with the Department of Electrical Engineering, Chalmers University of Technology, SE-41296 Gothenburg, Sweden (e-mail: {alexandre.graell, fredrik.brannstrom}@chalmers.se).

I. Andriyanova is with the ETIS-UMR8051 group, ENSEA/University of Cergy-Pontoise/CNRS, 95015 Cergy, France (e-mail: iryna.andriyanova@ensea.fr).

E. Rosnes is with Simula@UiB, N-5020 Bergen, Norway (e-mail: eirikrosnes@simula.no).

Piggyback codes presented in [19] are another class of codes that achieve a sub-optimal reduction in repair bandwidth with a much lower sub-packetization level in comparison to MSR codes, using the concept of *piggybacking*. Piggybacking consists of adding carefully chosen linear combinations of data symbols (called piggybacks) to the parity symbols of a given ECC. This results in a lower repair bandwidth at the expense of a higher complexity in encoding and repair operations. More recently, the authors in [20] presented a family of codes that reduce the encoding and repair complexity of PM-MBR codes while maintaining the same level of fault tolerance and repair bandwidth. However, this comes at the cost of large alphabet size. In [21], binary MDS array codes that achieve optimal repair bandwidth and low repair complexity were introduced, with the caveat that the file size is asymptotic and that the fault tolerance is limited to 3.

In this paper, we propose a family of non-MDS ECCs that achieve low repair bandwidth and low repair complexity while keeping the field size relatively small and having variable fault tolerance. In particular, we propose a systematic code construction based on two classes of parity symbols. Correspondingly, there are two classes of parity nodes. The first class of parity nodes, whose primary goal is to provide erasure correcting capability, is constructed using an MDS code modified by applying specially designed piggybacks to some of its code symbols. As a secondary goal, the first class of parity nodes enable to repair a number of data symbols at a low repair cost by downloading piggybacked symbols. The second class of parity nodes is constructed using a block code whose parity symbols are obtained through simple additions. The purpose of this class of parity nodes is not to enhance the erasure correcting capability, but rather to facilitate node repair at low repair bandwidth and low repair complexity by repairing the remaining failed symbols in the node. Compared to [22], we provide two constructions for the second class of parity nodes. The first one is given by a simple equation that represents the algorithmic construction in [22]. The second one is a heuristic construction that is more involved, but further reduces the repair bandwidth in some cases. Furthermore, we provide explicit formulas for the fault tolerance, repair bandwidth, and repair complexity of the proposed codes and numerically compare with other codes in the literature. The proposed codes achieve better repair bandwidth compared to MDS codes, Piggyback codes, generalized Piggyback codes [23], and exact-repairable MDS codes [24]. For certain code parameters, we also see that the proposed codes have better repair bandwidth compared to LRCs and Pyramid codes. Furthermore, they achieve better repair complexity than Zigzag codes, MDS codes, Piggyback codes, generalized Piggyback codes, exact-repairable MDS codes, and binary addition and shift implementable cyclic-convolutional (BASIC) PM-MBR codes [20]. Also, for certain code parameters, the codes have better repair complexity than Pyramid codes. The improvements over MDS codes, MSR codes, and the classes of Piggyback codes come at the expense of a lower fault tolerance in general.

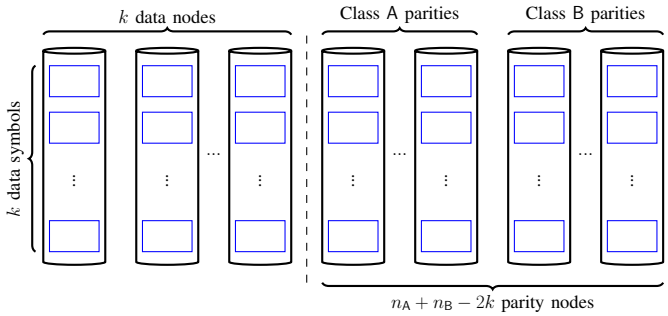

Fig. 1. System model of the DSS.

II. System Model and Code Construction

We consider the DSS depicted in Fig. 1, consisting of storage nodes, of which k are data nodes and $n-k$ are parity nodes. Consider a file that needs to be stored on the DSS. We represent a file as a $k \times k$ matrix $\boldsymbol{D} = [d_{i,j}]$, called the data array, over GF(q), where GF(q) denotes the Galois field of size q, with q being a prime number or a power of a prime number. In order to achieve reliability against node failures, the matrix $\boldsymbol{D}$ is encoded using an (n,k) vector code [25] to obtain a code matrix $\boldsymbol{C} = [c_{i,j}]$, referred to as the code array, of size $k \times n$, $c_{i,j} \in$ GF(q). The symbol $c_{i,j}$ in $\boldsymbol{C}$ is then stored at the i-th row of the j-th node in the DSS. Thus, each node stores k symbols. Each row in $\boldsymbol{C}$ is referred to as a stripe so that each file in the DSS is stored over k stripes in n storage nodes. We consider the (n,k) code to be systematic, which means that $c_{i,j} = d_{i,j}$ for $i,j = 0,\ldots,k-1$. Correspondingly, we refer to the k nodes storing systematic symbols as data nodes and the remaining $n-k$ nodes containing parity symbols only as parity nodes. The efficiency of the code is determined by the code rate, given by $R = k^2/kn = k/n$. Alternatively, the inverse of the code rate is referred to as the storage overhead.

For later use, we denote the set of message symbols in the k data nodes as $\mathcal{D} = \{d_{i,j}\}$ and by $\mathcal{P}_t$, $t = k,\ldots,n-1$, the set of parity symbols in the t-th node. Subsequently, we define the set $\mathcal{D}_{\mathcal{I}} \subseteq \mathcal{D}$ as

$$\mathcal{D}_{\mathcal{I}} = \{d_{i,j} \in \mathcal{D} \mid (i,j) \in \mathcal{I}\},$$

where $\mathcal{I}$ is an arbitrary index set. We also define the operator $(a+b)_k \triangleq (a+b) \bmod k$ for integers a and b.

Our main goal is to construct codes that yield low repair bandwidth and low repair complexity of a single failed data node. We focus on the repair of data nodes since the raw data is stored on these nodes and the users can readily access the data through these nodes. Thus, their survival is a crucial aspect of a DSS. To this purpose, we construct a family of systematic (n,k) codes consisting of two different classes of parity symbols. Correspondingly, there are two classes of parity nodes, referred to as Class A and Class B parity nodes, as shown in Fig. 1. Class A and Class B parity nodes are built using an (n_{A},k) code and an (n_{B},k) code, respectively, such that $n = n_{\mathsf{A}} + n_{\mathsf{B}} - k$. In other words, the parity nodes from the (n,k) code correspond to the parity nodes of Class A and Class B codes. The primary goal of Class A parity nodes is to achieve a good erasure correcting capability, while the purpose of Class B nodes is to yield low repair bandwidth and

low repair complexity. In particular, we focus on the repair of data nodes. The repair bandwidth (in bits) per node, denoted by γ, is proportional to the average number of symbols (data and parity) that need to be downloaded to repair a data symbol, denoted by λ. More precisely, let β be the sub-packetization level of the DSS, which is the number of symbols per node.[1] Then,

$$\lambda = \frac{\gamma}{\nu\beta}, \tag{1}$$

where $\nu = \lceil \log_2 q \rceil$ is the size (in bits) of a symbol. λ can be interpreted as the repair bandwidth normalized by the size (in bits) of a node, and will be referred to as the *normalized* repair bandwidth.

The main principle behind our code construction is the following. The repair is performed one symbol at a time. After the repair of a data symbol is accomplished, the symbols read to repair that symbol are cached in the memory. Therefore, they can be used to repair the remaining data symbols at no additional read cost. The proposed codes are constructed in such a way that the repair of a new data symbol requires a low additional read cost (defined as the number of additional symbols that need to be read to repair the data symbol), so that λ (and hence γ) is kept low.

Definition 1: The *read cost* of a symbol is the number of symbols that need to be read to repair the symbol. For a symbol that is repaired after some others, the *additional read cost* is defined as the number of additional symbols that need to be read to repair the symbol. (Note that symbols previously read to repair other data symbols are already cached in the memory and to repair a new symbol only some extra symbols may need to be read.)

III. Class A Parity Nodes

Class A parity nodes are constructed using a modified (n_{A}, k) MDS code, with $k+2 \le n_{\mathsf{A}} < 2k$, over GF($q$). In particular, we start from an (n_{A}, k) MDS code and apply piggybacks [19] to some of the parity symbols. The construction of Class A parity nodes is performed in two steps as follows.

1) Encode each row of the data array using an (n_{A}, k) MDS code (same code for each row). The parity symbol $p^{\mathsf{A}}_{i,j}$ is obtained as[2]

$$p^{\mathsf{A}}_{i,j} = \sum_{l=0}^{k-1} \alpha_{l,j} d_{i,l}, \ j = k, \ldots, n_{\mathsf{A}} - 1, \tag{2}$$

where $\alpha_{l,j}$ denotes a coefficient in GF(q) and $i = 0, \ldots, k-1$. Store the parity symbol in the corresponding row of the code array. Overall, $k(n_{\mathsf{A}}-k)$ parity symbols are generated.

2) Modify some of the parity symbols by adding piggybacks. Let τ, $1 \le \tau \le n_{\mathsf{A}} - k - 1$, be the number of piggybacks introduced per row. The parity symbol $p^{\mathsf{A}}_{i,u}$ is updated as

$$p^{\mathsf{A,p}}_{i,u} = p^{\mathsf{A}}_{i,u} + d_{(i+u-n_{\mathsf{A}}+\tau+1)_k, i}, \tag{3}$$

where $u = n_{\mathsf{A}} - \tau, \ldots, n_{\mathsf{A}} - 1$, the second term in the summation is the piggyback, and the superscript p in $p^{\mathsf{A,p}}_{i,u}$ indicates that the parity symbol contains piggybacks.

[1] For our code construction, $\beta = k$, but this is not the case in general.

[2] We use the superscript A to indicate that the parity symbol is stored in a Class A parity node.

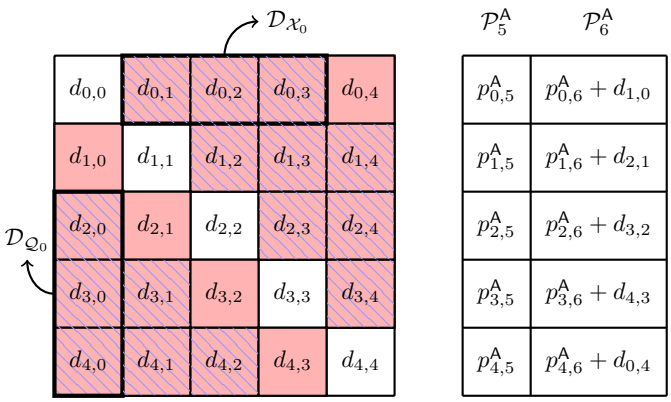

Fig. 2. A $(7,5)$ Class A code with $\tau = 1$ constructed from a $(7,5)$ MDS code. $\mathcal{P}^{\mathsf{A}}_5$ and $\mathcal{P}^{\mathsf{A}}_6$ are the parity nodes. For each row j, colored symbols belong to $\mathcal{D}_{\mathcal{R}_j}$.

The fault tolerance (i.e., the number of node failures that can be tolerated) of Class A codes is given in the following theorem.

Theorem 1: An (n_{A}, k) Class A code with τ piggybacks per row can tolerate

$$f = \begin{cases} n_{\mathsf{A}} - k - \tau + \left\lfloor \frac{\sqrt{(n_{\mathsf{A}}-k-\tau)^2+4k}-(n_{\mathsf{A}}-k-\tau)}{2} \right\rfloor & \text{if } \tau \ge \xi \\ n_{\mathsf{A}} - k & \text{if } \tau < \xi \end{cases}$$

node failures, where $\xi = \frac{\sqrt{(n_{\mathsf{A}}-k-\tau)^2+4k}-(n_{\mathsf{A}}-k-\tau)}{2}$.

Proof: See Appendix A. ∎

We remark that for $\tau < \xi$, Class A codes are MDS codes.

When a failure of a data node occurs, Class A parity nodes are used to repair $\tau+1$ of the k failed symbols. Class A parity symbols are constructed in such a way that, when node j is erased, $\tau+1$ data symbols in this node can be repaired reading the (non-failed) $k-1$ data symbols in the j-th row of the data array and $\tau+1$ parity symbols in the j-th row of Class A parity nodes (see also Section IV-C). For later use, we define the set $\mathcal{R}_j$ as follows.

Definition 2: For $j = 0, \ldots, k-1$, the index set $\mathcal{R}_j$ is defined as

$$\mathcal{R}_j = \{(j, (j+1)_k), (j, (j+2)_k), \ldots, (j, (j+k-1)_k)\}.$$

Then, the set $\mathcal{D}_{\mathcal{R}_j}$ is the set of $k-1$ data symbols that are read from row j to recover $\tau+1$ data symbols of node j using Class A parity nodes.

Example 1: An example of a Class A code is shown in Fig. 2. One can verify that the code can correct any 2 node failures. For each row j, the set $\mathcal{D}_{\mathcal{R}_j}$ is indicated in red color. For instance, $\mathcal{D}_{\mathcal{R}_0} = \{d_{0,1}, d_{0,2}, d_{0,3}, d_{0,4}\}$.

The main purpose of Class A parity nodes is to provide good erasure correcting capability. However, the use of piggybacks helps also in reducing the number of symbols that need to be read to repair the $\tau+1$ symbols of a failed node that are repaired using the Class A code, as compared to MDS codes.

The remaining $k-\tau-1$ data symbols of the failed node can also be recovered from Class A parity nodes, but at a high symbol read cost of k. Hence, the idea is to add another class of parity nodes, namely Class B parity nodes, in such a way that these symbols can be recovered with lower read cost.

IV. Class B Parity Nodes

Class B parity nodes are obtained using an (n_{B}, k) linear block code with $n_{\mathsf{B}} < 2k-\tau$ over GF(q) to encode the $k \times k$ data symbols of the data array. This generates $k(n_{\mathsf{B}}-k)$ Class B parity symbols, $p^{\mathsf{B}}_{i,l}$, $i = 0, \ldots, k-1$, $l = n_{\mathsf{A}}, \ldots, n-1$. In [22], we presented an algorithm to construct Class B codes. In this section, we present a similar construction in a much more compact, mathematical manner.

A. Definitions and Preliminaries

Definition 3: For $j = 0, \ldots, k-1$, the index set $\mathcal{Q}_j$ is defined as

$$\mathcal{Q}_j = \{((j+\tau+1)_k, j), ((j+\tau+2)_k, j), \ldots, ((j+k-1)_k, j)\}.$$

Assume that data node j fails. It is easy to see that the set $\mathcal{D}_{\mathcal{Q}_j}$ is the set of $k-\tau-1$ data symbols that are not recovered using Class A parity nodes.

Example 2: For the example in Fig. 2, the set $\mathcal{D}_{\mathcal{Q}_j}$ is indicated by hatched symbols for each column j, $j = 0, \ldots, k-1$. For instance, $\mathcal{D}_{\mathcal{Q}_0} = \{d_{2,0}, d_{3,0}, d_{4,0}\}$.

For later use, we also define the following set.

Definition 4: For $j = 0, \ldots, k-1$, the index set $\mathcal{X}_j$ is defined as

$$\mathcal{X}_j = \{(j, (j+1)_k), (j, (j+2)_k), \ldots, (j, (j+k-\tau-1)_k)\}.$$

Note that $\mathcal{X}_j = \mathcal{R}_j \cap \{\cup_{l=0}^{k-1} \mathcal{Q}_l\}$.

Example 3: For the example in Fig. 2, the set $\mathcal{D}_{\mathcal{X}_j}$ is indicated by hatched symbols for each row j. For instance, $\mathcal{X}_0 = \mathcal{R}_0 \cap \{\mathcal{Q}_0 \cup \mathcal{Q}_1 \cup \mathcal{Q}_2 \cup \mathcal{Q}_3 \cup \mathcal{Q}_4\} = \{(0,1), (0,2), (0,3)\}$, thus we have $\mathcal{D}_{\mathcal{X}_0} = \{d_{0,1}, d_{0,2}, d_{0,3}\}$.

The purpose of Class B parity nodes is to allow the recovery of the data symbols in $\mathcal{D}_{\mathcal{Q}_j}$, $j = 0, \ldots, k-1$, at a low additional read cost. Note that after recovering $\tau+1$ symbols using Class A parity nodes, the data symbols in the sets $\mathcal{D}_{\mathcal{R}_j}$ are already stored in the decoder memory. Therefore, they are accessible for the recovery of the remaining $k-\tau-1$ data symbols using Class B parity nodes without the need of reading them again. The main idea is based on the following proposition.

Proposition 1: If a Class B parity symbol p^{B} is the sum of one data symbol $d \in \mathcal{D}_{\mathcal{Q}_j}$ and a number of data symbols in $\mathcal{D}_{\mathcal{X}_j}$, then the recovery of d comes at the cost of one additional read (one should read parity symbol p^{B}).

This observation is used in the construction of Class B parity nodes in Section IV-B below to reduce the normalized repair bandwidth λ. In particular, we add up to $k-\tau-1$ Class B parity nodes which allow to reduce the additional read cost of all $k(k-\tau-1)$ data symbols in all $\mathcal{D}_{\mathcal{Q}_j}$'s to 1. (The addition of a single Class B parity node allows to recover one new data symbol in each $\mathcal{D}_{\mathcal{Q}_j}$, $j = 0, \ldots, k-1$, at the cost of one additional read.)

$\mathcal{P}_7^{\mathsf{B}}$	$\mathcal{P}_8^{\mathsf{B}}$	$\mathcal{P}_9^{\mathsf{B}}$
$d_{2,0}+d_{0,1}+d_{0,2}$	$d_{3,0}+d_{0,1}$	$d_{4,0}$
$d_{3,1}+d_{1,2}+d_{1,3}$	$d_{4,1}+d_{1,2}$	$d_{0,1}$
$d_{4,2}+d_{2,3}+d_{2,4}$	$d_{0,2}+d_{2,3}$	$d_{1,2}$
$d_{0,3}+d_{3,4}+d_{3,0}$	$d_{1,3}+d_{3,4}$	$d_{2,3}$
$d_{1,4}+d_{4,0}+d_{4,1}$	$d_{2,4}+d_{4,0}$	$d_{3,4}$

Fig. 3. Class B parity nodes for the data nodes in Fig. 2.

B. Construction of Class B Nodes

For $t = 0, \ldots, k-1$, each parity symbol in the l-th Class B parity node, $l = n_{\mathsf{A}}, \ldots, n-1$, is sequentially constructed as

$$p^{\mathsf{B}}_{t,l} = d_{(\tau+1-n_{\mathsf{A}}+l+t)_k, t} + \sum_{j=0}^{k-\tau-3+n_{\mathsf{A}}-l} d_{t,(1+j+t)_k}. \quad (4)$$

The construction above follows Proposition 1 wherein $d_{(\tau+1-n_{\mathsf{A}}+l+t)_k, t} \in \mathcal{D}_{\mathcal{Q}_t}$ and $\{d_{t,(1+j+t)_k}\}_{j=0}^{k-\tau-3+n_{\mathsf{A}}-l} \subset \mathcal{D}_{\mathcal{X}_t}$. This ensures that the read cost of each of the k symbols $d_{(\tau+1-n_{\mathsf{A}}+l+t)_k, t}$ is 1. Thus, the addition of each parity node leads to k data symbols to have a read cost of 1. Note that adding the second term in (4) ensures that $k(k-\tau-1)$ data symbols are repaired by the Class B parity nodes. The same principle was used in [22]. It should be noted that the set of data symbols used in the construction of the parity symbols in (4) may be different compared to the construction in [22]. However, the overall average repair bandwidth remains the same.

Remark 1: For the particular case $n_{\mathsf{B}}-k = k-\tau-1$ one may neglect the second term in (4). The resulting codes would still have the same repair bandwidth and lower repair complexity than the codes built from (4). However, this construction would not allow rate-compatible Class B codes.

In the sequel, we will refer to the construction of Class B parity nodes according to (4) as Construction 1.

Example 4: With the aim to construct a $(10, 5)$ code, consider the construction of an $(8, 5)$ Class B code where the $(7, 5)$ Class A code, with $\tau = 1$, is as shown in Fig. 2. For $t = 0, \ldots, k-1$, the parity symbols in the first Class B parity node (the 7-th node) are

$$p^{\mathsf{B}}_{t,7} = d_{(2+t)_5, t} + \sum_{j=0}^{1} d_{t,(1+j+t)_5} = d_{(2+t)_5, t} + d_{t,(1+t)_5} + d_{t,(2+t)_5}.$$

The constructed parity symbols are as seen in Fig. 3, where the t-th row in node $\mathcal{P}_7^{\mathsf{B}}$ contains the parity symbol $p^{\mathsf{B}}_{t,7}$. Notice that $d_{(2+t)_5, t} \in \mathcal{D}_{\mathcal{Q}_t}$ and $\{d_{t,(1+t)_5}, d_{t,(2+t)_5}\} \subset \mathcal{D}_{\mathcal{X}_t}$. In a similar way, the parity symbols in nodes $\mathcal{P}_8^{\mathsf{B}}$ and $\mathcal{P}_9^{\mathsf{B}}$ are

$$p^{\mathsf{B}}_{t,8} = d_{(3+t)_5, t} + \sum_{j=0}^{0} d_{t,(1+j+t)_5} = d_{(3+t)_5, t} + d_{t,(1+t)_5}$$

and

$$p^{\mathsf{B}}_{t,9} = d_{(4+t)_5,t} + \sum_{j=0}^{-1} d_{t,(1+j+t)_5} = d_{(4+t)_5,t},$$

respectively.

Consider the repair of the first data node in Fig. 2. The symbol $d_{0,0}$ is reconstructed using $p^{\mathsf{A}}_{0,5}$. This requires reading the symbols $d_{0,1}$, $d_{0,2}$, $d_{0,3}$, and $d_{0,4}$. Since $p^{\mathsf{A}}_{0,6}$ is a function of all data symbols in the first row, reading $p^{\mathsf{A}}_{0,6} + d_{1,0}$ is sufficient for the recovery of $d_{1,0}$. From Fig. 3, the symbols $d_{2,0}$, $d_{3,0}$, and $d_{4,0}$ can be recovered by reading just the parities $d_{2,0} + d_{0,1} + d_{0,2}$, $d_{3,0} + d_{0,1}$, and $d_{4,0}$, respectively. Thus, reading $5 + 4 = 9$ symbols is sufficient to recover all the symbols in the node, and the normalized repair bandwidth is $9/5 = 1.8$ per failed symbol. A more formal repair procedure is presented in Section IV-C.

Adding $n_{\mathsf{B}} - k$ Class B parity nodes allows to reduce the additional read cost of $n_{\mathsf{B}} - k$ data symbols from each $\mathcal{D}_{\mathcal{Q}_j}$, $j = 0, \ldots, k-1$, to 1. However, this comes at the cost of a reduction in the code rate, i.e., the storage overhead is increased. In the above example, adding $n_{\mathsf{B}} - k = 3$ Class B parity nodes leads to the reduction in code rate from $R = 5/7$ to $R = 5/10 = 1/2$. If a lower storage overhead is required, Class B parity nodes can be *punctured*, starting from the last parity node (for the code in Example 4, nodes $\mathcal{P}^{\mathsf{B}}_9$, $\mathcal{P}^{\mathsf{B}}_8$, and $\mathcal{P}^{\mathsf{B}}_7$ can be punctured in this order), at the expense of an increased repair bandwidth. If all Class B parity nodes are punctured, only Class A parity nodes would remain, and the repair bandwidth is equal to the one of the Class A code. Thus, our code construction gives a family of rate-compatible codes which provides a tradeoff between repair bandwidth and storage overhead: adding more Class B parity nodes reduces the repair bandwidth, but also increases the storage overhead.

C. Repair of a Single Data Node Failure: Decoding Schedule

The repair of a failed data node proceeds as follows. First, $\tau + 1$ symbols are repaired using Class A parity nodes. Then, the remaining symbols are repaired using Class B parity nodes. With a slight abuse of language, we will refer to the repair of symbols using Class A and Class B parity nodes as the decoding of Class A and Class B codes, respectively.

We will need the following definition.

Definition 5: Consider a Class B parity node and let $\mathcal{P}^{\mathsf{B}}$ denote the set of parity symbols in this node. Also, let $d \in \mathcal{D}_{\mathcal{Q}_j}$ for some j and $p^{\mathsf{B}} \in \mathcal{P}^{\mathsf{B}}$ be the parity symbol $p^{\mathsf{B}} = d + \sum_{d' \in \mathcal{D}'} d'$, where $\mathcal{D}' \subset \mathcal{D}$, i.e., the parity symbol p^{B} is the sum of d and a subset of other data symbols. We define $\breve{\mathcal{D}} = \mathcal{D}' \cup \{d\}$.

Suppose that node j fails. Decoding is as follows.

- **Decoding the Class A code**. To reconstruct the failed data symbol in the j-th row of the code array, k symbols ($k-1$ data symbols and $p^{\mathsf{A}}_{j,k}$) in the j-th row are read. These symbols are now cached in the memory. We then read the τ piggybacked symbols in the j-th row. By construction (see (3)), this allows to repair τ failed symbols, at the cost of an additional read each.
- **Decoding the Class B code**. Each remaining failed data symbol $d_{i,j} \in \mathcal{D}_{\mathcal{Q}_j}$ is obtained by reading a Class B parity symbol whose corresponding set $\breve{\mathcal{D}}$ (see Definition 5) contains $d_{i,j}$. In particular, if several Class B parity symbols $p^{\mathsf{B}}_{i',j'}$ contain $d_{i,j}$, we read the parity symbol with largest index j'. This yields the lowest additional read cost.

V. A Heuristic Construction of Class B Nodes With Improved Repair Bandwidth

In this section, we provide a way to improve the repair bandwidth of the family of codes constructed so far. More specifically, we achieve this by providing a heuristic algorithm for the construction of the Class B code, which improves Construction 1 in Section IV for some values of n and even values of k.

The algorithm is based on a simple observation. Let p^{B}_1 and p^{B}_2 be two parity symbols constructed from ρ data symbols in $\mathcal{D}$ in two different ways as follows:

$$p^{\mathsf{B}}_1 = d_{i,j} + d_{j,i} + d_{j,i_2} + \cdots + d_{j,i_{\rho-1}}, \tag{5}$$

$$p^{\mathsf{B}}_2 = d_{i,j} + d_{j,i_1} + d_{j,i_2} + \cdots + d_{j,i_{\rho-1}}, \tag{6}$$

where $d_{i,j} \in \mathcal{D}_{\mathcal{Q}_j}$ (see Definition 3), $i_1, \ldots, i_{\rho-1} \neq i$, and $d_{j,i_1}, d_{j,i_2}, \ldots, d_{j,i_{\rho-1}} \in \mathcal{D}_{\mathcal{X}_j}$ (see Definition 4). Note that the only difference between the two parity symbols above is that p^{B}_2 does not involve $d_{j,i}$ (and that p^{B}_1 does not involve d_{j,i_1}). This has a major consequence in the repair of the data symbols $d_{i,j}, d_{j,i}, \ldots, d_{j,i_\rho-1}$ and $d_{i,j}, d_{j,i_1}, \ldots, d_{j,i_\rho-1}$ using p^{B}_1 and p^{B}_2, respectively. Consider the repair using parity symbol p^{B}_1. From Proposition 1, it is clear that the repair of symbol $d_{i,j}$ will have an additional read cost of 1, since the remaining $\rho - 1$ data symbols are in $\mathcal{D}_{\mathcal{X}_j}$. As the symbol $d_{j,i} \in \mathcal{D}_{\mathcal{Q}_i}$ and $d_{i,j} \in \mathcal{D}_{\mathcal{X}_i}$, from Proposition 1 and the fact that $d_{j,i_2}, \ldots, d_{j,i_{\rho-1}} \notin \mathcal{D}_{\mathcal{X}_i}$, we can repair $d_{j,i}$ with an additional read cost of $\rho - 1$. The remaining $\rho - 2$ symbols each have an additional read cost of ρ, whereas the symbols repaired using p^{B}_2 incur an additional read cost of 1 for the symbol $d_{i,j}$ and ρ for the remaining symbols. Clearly, we see that the combined additional read cost, i.e., the sum of the individual additional read costs for each data symbol using p^{B}_1 is lower (by 1) than that using p^{B}_2.

In the way Class A parity nodes are constructed and due to the structure of the sets $\mathcal{D}_{\mathcal{Q}_j}$ and $\mathcal{D}_{\mathcal{X}_j}$, it can be seen that $d_{i,j} \in \mathcal{D}_{\mathcal{Q}_j}$ and $d_{j,i} \in \mathcal{D}_{\mathcal{X}_j}$ when $k \geq 2(\tau+1)$. From Construction 1 of the Class B code in Section IV we observe that for odd k and $k > 2(\tau+1)$, the parity symbols in node $\mathcal{P}^{\mathsf{B}}_l$ are as in (5) for $n_{\mathsf{A}} \leq l \leq n_{\mathsf{A}} + \lfloor k/2 \rfloor - \tau - 1$. Furthermore, for $n_{\mathsf{A}} + \lfloor k/2 \rfloor - \tau \leq l \leq n-1$, the parity symbols in node $\mathcal{P}^{\mathsf{B}}_l$ have the structure in (6). On the other hand, for k even and $k \geq 2(\tau+1)$, the parity symbols in the node $\mathcal{P}^{\mathsf{B}}_l$ are as in (5) for $n_{\mathsf{A}} \leq l \leq n_{\mathsf{A}} + k/2 - \tau - 2$. However, contrary to case of k odd, the parity symbols in the node $\mathcal{P}^{\mathsf{B}}_{n_{\mathsf{A}}+k/2-\tau-1}$ follow (6). But since $k \geq 2(\tau+1)$, we know that $d_{i,j} \in \mathcal{D}_{\mathcal{Q}_j}$ and $d_{j,i} \in \mathcal{D}_{\mathcal{X}_j}$. Thus, it is possible to construct some parity symbols in this node as in (5), and Construction 1 of Class B nodes in the previous section can be improved. However, the improvement

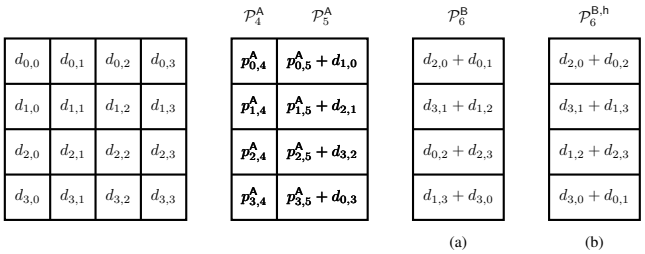

Fig. 4. A $(7, 4)$ code constructed from a $(6, 4)$ Class A code with $\tau = 1$ and a $(5, 4)$ Class B code. (a) Class B node constructed according to Construction 1 in Section IV. (b) A different configuration of the Class B node that reduces the repair bandwidth.

comes at the expense of the loss of the mathematical structure of Class B nodes given in (4).

Example 5: Consider the $(7, 4)$ code as shown in Fig. 4. Fig. 4(a) shows the node $\mathcal{P}_6^{\mathsf{B}}$ using Construction 1 in Section IV, while Fig. 4(b) shows a different configuration of the node $\mathcal{P}_6^{\mathsf{B}}$. Note that $k = 2(\tau + 1) = 4$. Thus, each pair $(\mathcal{D}_{\mathcal{Q}_j}, \mathcal{D}_{\mathcal{X}_j})$ contains one symbol $d_{i,j}$ and $d_{j,i}$. The node $\mathcal{P}_6^{\mathsf{B}}$ has parity symbols according to (6), while $\mathcal{P}_6^{\mathsf{B,h}}$ has two parity symbols as in (5) and two parity symbols according to (6). The configuration of the $(7, 4)$ code arising from Construction 1 has a normalized repair bandwidth of 2, while the $(7, 4)$ code with node $\mathcal{P}_6^{\mathsf{B,h}}$ in Fig. 4(b) has a repair bandwidth of 1.825, i.e., an improvement is achieved.

In order to describe the modified code construction, we define the function $\mathrm{read}(d, p^{\mathsf{B}})$ as follows.

Definition 6: Consider the construction of the parity symbol p^{B} as $p^{\mathsf{B}} = d + \sum_{d' \in \breve{\mathcal{D}}} d'$ (see Definition 5). Then,

$$\mathrm{read}(d, p^{\mathsf{B}}) = |\breve{\mathcal{D}} \backslash \mathcal{D}_{\mathcal{X}_j}|.$$

For a given data symbol d, the function $\mathrm{read}(d, p^{\mathsf{B}})$ gives the additional number of symbols that need to be read to recover d (considering the fact that some symbols are already cached in the memory). The set $\breve{\mathcal{D}}$ represents the set of data symbols that the parity symbol p^{B} is a function of. We use the index set $\mathcal{U}$ to represent the indices of such data symbols. We denote by $\mathcal{U}_t, t = 0, \ldots, k-1$, the index set corresponding to the t-th parity symbol in the node (there are k parity symbols in a parity node).

In the following, denote by $\boldsymbol{A} = [a_{i,j}]$ a temporary matrix of read costs for the respective data symbols in $\boldsymbol{D} = [d_{i,j}]$. After Class A decoding,

$$a_{i,j} = \begin{cases} \infty & \text{if } d_{i,j} \in \cup_{t=0}^{k-1} \mathcal{D}_{\mathcal{Q}_t} \\ k & \text{if } i = j \\ 1 & \text{otherwise} \end{cases}. \tag{7}$$

In Section V-A below, we will show that the construction of parities depends upon the entries of $\boldsymbol{A}$. To this extent, for some real matrix $\boldsymbol{M} = [m_{i,j}]$ and index set $\mathcal{I}$, we define $\Psi(\boldsymbol{M}_{\mathcal{I}})$ as the set of indices of matrix elements of $\boldsymbol{M}$ from $\mathcal{I}$ whose values are equal to the maximum of all entries in $\boldsymbol{M}$ indexed by $\mathcal{I}$. More formally, $\Psi(\boldsymbol{M}_{\mathcal{I}})$ is defined as

$$\Psi(\boldsymbol{M}_{\mathcal{I}}) = \left\{ (i, j) \in \mathcal{I} \mid m_{i,j} = \max_{(i',j') \in \mathcal{I}} m_{i',j'} \right\}.$$

The heuristic algorithm to construct the Class B code is given in Appendix B and we will refer to the construction of the Class B code according to this algorithm as Construction 2. In the following subsection, we clarify the heuristic algorithm to construct the Class B code with the help of a simple example.

A. Construction Example

Let us consider the construction of a $(7, 4)$ code using a $(6, 4)$ Class A code and a $(5, 4)$ Class B code. In total, there are three parity nodes; two Class A parity nodes, denoted by $\mathcal{P}_4^{\mathsf{A}}$ and $\mathcal{P}_5^{\mathsf{A}}$, respectively, and one Class B parity node, denoted by $\mathcal{P}_6^{\mathsf{B,h}}$, where the upper index h is used to denote that the parity node is constructed using the heuristic algorithm. The parity symbols of the nodes are depicted in Fig. 4. Each parity symbol of the Class B parity node is the sum of $k - \tau - 1$ data symbols $d_{i,j} \in \cup_{j'} \mathcal{D}_{\mathcal{Q}_{j'}}$, constructed such that the read cost of each symbol $d_{i,j}$ is lower than $a_{i,j}$ as shown below.

1. Construction of $\mathcal{P}_6^{\mathsf{B,h}}$

Each parity symbol in this node is constructed using $\rho_6 = k - \tau - 1 = 2$ unique symbols as follows.

- 1.a Since no symbols have been constructed yet, we have $\mathcal{U}_{t_1} = \emptyset$, $t_1 = 0, \ldots, 3$. (This corresponds to the execution of Line 1 to Line 19 of Algorithm 2 in Appendix B.)
- 1.b Select $d_{i,0} \in \mathcal{D}_{\mathcal{Q}_0}$ such that its read cost is maximum, i.e., $d_{i,0} \in \mathcal{D}_{\Psi(\boldsymbol{A}_{\mathcal{Q}_0})}$. Choose $d_{i,0} = d_{2,0}$, as $a_{2,0} = \infty$. Note that we choose $d_{2,0}$ since $d_{0,2} \in \mathcal{D}_{\mathcal{X}_0}$.
- 1.c Construct $p_{0,6} = d_{2,0} + d_{0,2}$ (see Line 4 of Algorithm 2). Correspondingly, we have $\mathcal{U}_0 = \{(2, 0), (0, 2)\}$.
- 1.d Recursively construct the next parity symbol in the node as follows. Similar to Item 1.b, choose $d_{3,1} \in \mathcal{D}_{\Psi(\boldsymbol{A}_{\mathcal{Q}_1})}$. Construct $p_{1,6} = d_{3,1} + d_{1,3}$. Likewise, we have $\mathcal{U}_1 = \{(3, 1), (1, 3)\}$
- 1.e For the next parity symbol, note that $d_{0,2}$ is already used in the construction of $p_{0,6}$. The only possible choice of symbol in $\mathcal{D}_{\mathcal{Q}_2}$ is $d_{1,2}$, but $d_{2,1} \notin \mathcal{D}_{\mathcal{X}_2}$. Therefore, we choose $d_{i_1,2} \in \mathcal{D}_{\Psi(\boldsymbol{A}_{\mathcal{Q}_2 \backslash \cup_{j'} \mathcal{U}_{j'}})}$ (see Line 7 of Algorithm 2). In particular, since $a_{1,2} = \infty$, we choose $d_{i_1,2} = d_{1,2}$. Then, Lines 8 to 11 of Algorithm 2 are executed.
- 1.f Choose an element $d_{2,i_2} \in \mathcal{D}_{\mathcal{X}_2 \backslash \cup_{j'} \mathcal{U}_{j'}}$. In other words choose a symbol in $\mathcal{D}_{\mathcal{X}_2}$ which has not been used in $p_{0,6}$ and $p_{1,6}$. We have $d_{2,i_2} = d_{2,3}$. Construct $p_{2,6} = d_{1,2} + d_{2,3}$. Thus, $\mathcal{U}_2 = \{(1, 2), (2, 3)\}$.
- 1.g To construct the last parity symbol, we look for data symbols from the sets $\mathcal{D}_{\mathcal{Q}_3}$ and $\mathcal{D}_{\mathcal{X}_3}$. However, all symbols in $\mathcal{D}_{\mathcal{Q}_3}$ have been used in the construction of previous parity symbols. Therefore, we cyclically shift to the next pair of sets $(\mathcal{D}_{\mathcal{Q}_0}, \mathcal{D}_{\mathcal{X}_0})$. Following Items 1.e and 1.f, we have $p_{3,6} = d_{3,0} + d_{0,1}$ and $\mathcal{U}_3 = \{(3, 0), (0, 1)\}$.

Note that $|\mathcal{U}_t| = 2$ for all t, thus this completes the construction of the $(7, 4)$ code. The Class B parity node constructed above is depicted in Fig. 4(b).

B. Discussion

In general, the algorithm constructs $n_{\mathsf{B}} - k$ parity nodes, $\mathcal{P}^{\mathsf{B}}_{n_{\mathsf{A}}}, \ldots, \mathcal{P}^{\mathsf{B}}_{n-1}$, recursively. In the l-th Class B node, $l = n_{\mathsf{A}}, \ldots, n-1$, each parity symbol is a sum of at most $\rho_l = k - \tau - 1 - l + n_{\mathsf{A}}$ symbols $d_{i,j} \in \cup_{j'} \mathcal{D}_{\mathcal{Q}_{j'}}$. Each parity symbol $p_{t,l}$, $t = 0, \ldots, k-1$, in the l-th Class B parity node with $\rho_l > 1$ is constructed recursively with $\rho_l - 1$ recursion steps. In the first recursion step, each parity symbol $p_{t,l}$ is either equal to a single data symbol or a sum of 2 data symbols. In the latter case, the first symbol $d_{i,j} \in \mathcal{D}_{\mathcal{Q}_t}$ is chosen as the symbol with the largest read cost $a_{i,j}$. The second symbol is $d_{j,i} \in \mathcal{D}_{\mathcal{X}_t}$ if such a symbol exists. Otherwise (i.e., if $d_{j,i} \notin \mathcal{D}_{\mathcal{X}_t}$), symbol $d_{j,i''} \in \mathcal{D}_{\mathcal{X}_t}$ is chosen. In the remaining $\rho_l - 2$ recursion steps a subsequent data symbol $d_{j,i'} \in \mathcal{D}_{\mathcal{X}_t}$ (if it exists) is added to $p_{t,l}$. Doing so ensures that k symbols have a new read cost that is reduced to 1 when parity symbols $p_{t,l}$ are used to recover them. Having obtained these parity symbols, the read costs of all data symbols in $\cup_{j'} \mathcal{D}_{\mathcal{Q}_{j'}}$ are updated and stored in $\boldsymbol{A}$. This process is repeated for successive parity nodes. If $\rho_l = 1$ for the l-th parity node, its parity symbols $p_{t,l}$ are equal to the data symbols $d_{i,j} \in \mathcal{D}_{\mathcal{Q}_t}$ whose read costs $a_{i,j}$ are the maximum possible.

In the above example, only a single recursion for the construction of $\mathcal{P}^{\mathsf{B},\mathsf{h}}_6$ is needed, where each parity symbol is a sum of two data symbols.

VI. Code Characteristics and Comparison

In this section, we characterize different properties of the codes presented in Sections III-V. In particular, we focus on the fault tolerance, repair bandwidth, repair complexity, and encoding complexity. We consider the complexity of elementary arithmetic operations on field elements of size $\nu = m\lceil \log_2 p \rceil$ in GF(q), where $q = p^m$ for some prime number p and positive integer m. The complexity of addition is $O(\nu)$, while that of multiplication is $O(\nu^2)$, where the argument of $O(\cdot)$ denotes the number of elementary binary additions.[3]

A. Code Rate

The code rate for the proposed codes is given by $R = \frac{k}{n_{\mathsf{A}} + n_{\mathsf{B}} - k}$. It can be seen that the code rate is bounded as

$$\frac{k}{3k - \tau - 2} \leq R \leq \frac{k}{k+3}.$$

The upper bound is achieved when $n_{\mathsf{A}} = k+2$, $\tau = 1$, and $n_{\mathsf{B}} = k+1$, while the lower bound is obtained from the upper bounds on n_{A} and n_{B} given in Sections III and IV.

B. Fault Tolerance

The proposed codes have fault tolerance equal to that of the corresponding Class A codes, which depends on the MDS code used in their construction and τ (see Theorem 1). Class B nodes do not help in improving the fault tolerance. The reason is that improving the fault tolerance of the Class A code requires the knowledge of the piggybacks that are strictly not in the set $\cup_j \mathcal{D}_{Q_j}$, while Class B nodes can only be used to repair symbols in $\cup_j \mathcal{D}_{Q_j}$.

In the case where the Class B code has parameters $(n_{\mathsf{B}} = k+1, k)$, the resulting $(n = n_{\mathsf{A}}+1, k)$ code has fault tolerance $n_{\mathsf{A}} - k$ for $\tau < \xi$, i.e., one less than that of an $(n = n_{\mathsf{A}}+1, k)$ MDS code.

C. Repair Bandwidth of Data Nodes

According to Section IV-C, to repair the first $\tau+1$ symbols in a failed node, $k-1$ data symbols and $\tau+1$ Class A parity symbols are read. The remaining $k-\tau-1$ data symbols in the failed node are repaired by reading Class B parity symbols.

Let f_l, $l = n_{\mathsf{A}}, \ldots, n-1$, denote the number of parity symbols that are used from the l-th Class B node according to the decoding schedule in Section IV-C. Due to Construction 1 in Section IV, we have

$$\begin{aligned} f_{n_{\mathsf{A}}} = f_{n_{\mathsf{A}}+1} = \cdots = f_{n-2} = 1, \\ f_{n-1} = k - \tau - 1 - \sum_{l=n_{\mathsf{A}}}^{n-2} f_l = k - \tau - n + n_{\mathsf{A}}. \end{aligned} \tag{8}$$

The Class B nodes $n_{\mathsf{A}}, \ldots, n-2$ are used to repair $n-1-n_{\mathsf{A}}$ symbols with an additional read cost of $n-1-n_{\mathsf{A}}$ (1 per symbol). The remaining $k-\tau-n+n_{\mathsf{A}} = f_{n-1}$ erased symbols are corrected using the $(n-1)$-th Class B node. The repair of one of the f_{n-1} symbols entails an additional read cost of 1. On the other hand, since the parity symbols in the $(n-1)$-th Class B node are a function of f_{n-1} symbols, the repair of the remaining $f_{n-1}-1$ symbols entails an additional read cost of at most f_{n-1} each. In all, the $k-\tau-1$ erased symbols in the failed node have a total additional read cost of at most $n - n_{\mathsf{A}} + (f_{n-1}-1)f_{n-1}$. The normalized repair bandwidth for the failed systematic node is therefore given as

$$\lambda^{\mathsf{s}} \leq \frac{k + \tau + n - n_{\mathsf{A}} + (f_{n-1}-1)f_{n-1}}{k} = \frac{2k - 2f_{n-1} + f_{n-1}^2}{k}.$$

Note that f_{n-1} is function of τ, and it follows from Section VI-B and (8) that when τ increases, the fault tolerance reduces while λ^{s} improves. Furthermore, as n_{B} increases (thereby as n increases), f_{n-1} decreases. This leads to a further reduction of the normalized repair bandwidth.

D. Repair Complexity of a Failed Data Node

To repair the first symbol requires k multiplications and $k-1$ additions. To repair the following τ symbols require an additional τk multiplications and additions. Thus, the repair complexity of repairing $\tau+1$ failed symbols is

$$C^{\mathsf{A}}_{\mathsf{r}} = O((k-1)\nu + k\nu^2) + O(\tau k(\nu + \nu^2)).$$

For Construction 1, the remaining $k-\tau-1$ failed data symbols in the failed node are corrected using $k-\tau-1$ parity symbols from $n_{\mathsf{B}} - k$ Class B nodes. To this extent, note that

[3] It should be noted that the complexity of multiplication is quite pessimistic. However, for the sake of simplicity we assume it to be $O(\nu^2)$. When the field is GF(2^ν) there exist algorithms such as the Karatsuba-Ofman algorithm [26], [27] and the Fast Fourier Transform [28]–[30] that lower the complexity to $O(\nu^{\log_2 3})$ and $O(\nu \log_2 \nu)$, respectively.

TABLE I

COMPARISON OF (n,k) CODES THAT AIM AT REDUCING THE REPAIR BANDWIDTH. THE REPAIR BANDWIDTH AND THE REPAIR COMPLEXITY ARE NORMALIZED PER SYMBOL, WHILE THE ENCODING COMPLEXITY IS GIVEN PER ROW OF THE CODE ARRAY. NOTE THAT FOR MDR AND EVENODD CODES, $n = k+2$ WHERE k IS PRIME FOR EVENODD CODES.

	β	**Fault Tolerance**	**Norm. Repair Band.**	**Norm. Repair Compl.**	**Enc. Complexity**
MDS	1	$n-k$	k	$O((k-1)\nu + k\nu^2)$	$O((n-k)(k-1)\nu + k\nu^2)$
LRC [3]	1	$r+1$	$\frac{k}{n-k-r}$	$O((\lceil \frac{k}{n-k-r} \rceil - 1)\nu)$	$rO((k-1)\nu + k\nu^2) + (n-k-r)O((\lceil \frac{k}{n-k-r} \rceil - 1)\nu)$
MDR [9]	2^k	2	$\frac{k+1}{2}$	$O((k-1)\nu)$	$O(2(k-1)\nu)$
Zigzag [10]	$(n-k)^{k-1}$	$n-k$	$\frac{n-1}{n-k}$	$O((k-1)\nu + k\nu^2)$	$O((n-k)(k-1)\nu + k\nu^2)$
Piggyback [19]	2	$n-k$	$\frac{(k-t_r)(k+t)+t_r(k+t_r+\ell-2)}{2k}$	–	–
EVENODD [25]	$k-1$	2	k	$O((k-1)\nu)$	$O(\frac{2k^2-2k-1}{k-1}\nu)$
Proposed codes	k	f	λ^{s}	$C_{\mathsf{r}}^{\mathsf{s}}/k$	$C_{\mathsf{e}}^{\mathsf{s}}$

$\sum_{l=n_{\mathsf{A}}}^{n-1} f_l = k-\tau-1$. The repair complexity for repairing the remaining $k-\tau-1$ symbols is

$$C_{\mathsf{r}}^{\mathsf{B}} = \sum_{l=n_{\mathsf{A}}}^{n-1} O(f_l(k-\tau-2-l+n_{\mathsf{A}})\nu). \tag{9}$$

From (8), (9) simplifies to

$$\begin{aligned} C_{\mathsf{r}}^{\mathsf{B}} &= \sum_{l=n_{\mathsf{A}}}^{n-2} O((k-\tau-2-l+n_{\mathsf{A}})\nu) \\ &\quad + O((k-\tau-n+n_{\mathsf{A}})(k-\tau-1-n+n_{\mathsf{A}})\nu) \\ &= O\bigg(\frac{1}{2}(n-1-n_{\mathsf{A}})(2k-2\tau-2+n_{\mathsf{A}}-n)\nu\bigg) \\ &\quad + O((k-\tau-n+n_{\mathsf{A}})(k-\tau-1-n+n_{\mathsf{A}})\nu). \end{aligned}$$

For Construction 2, the final $k-\tau-1$ failed data symbols require at most $k-\tau-2$ additions, since Class B parity symbols are constructed as sums of at most $k-\tau-1$ data symbols. The corresponding repair complexity is therefore

$$C_{\mathsf{r}}^{\mathsf{B}} \leq O((k-\tau-2)(k-\tau-1)\nu).$$

Finally, the total repair complexity is $C_{\mathsf{r}}^{\mathsf{s}} = C_{\mathsf{r}}^{\mathsf{A}} + C_{\mathsf{r}}^{\mathsf{B}}$.

E. Repair Bandwidth and Complexity of Parity Nodes

We characterize the normalized repair bandwidth and repair complexity of Class A and B parity nodes.

Class A nodes consist of $n_{\mathsf{A}} - k$ MDS parity nodes of which τ nodes are modified with a single piggyback. Thus, the repair of each parity symbol in the $n_{\mathsf{A}} - k - \tau$ non-modified nodes requires downloading k data symbols. To obtain the parity symbol, one needs to perform $k-1$ additions and k multiplications. Thus, each parity symbol in these nodes has a repair bandwidth of k and a repair complexity of $O((k-1)\nu + k\nu^2)$, while each erased parity symbol in the τ piggybacked nodes requires reading $k+1$ data symbols. Such parity symbols are obtained by performing $k-1$ additions and k multiplications to get the original MDS parity symbol and then finally a single addition of the piggyback to the MDS parity symbol is required. Overall, the normalized repair bandwidth is $k+1$ and the normalized repair complexity is $O(k(\nu+\nu^2))$. In average, the normalized repair bandwidth and the normalized repair complexity of Class A parity nodes are

$$\begin{aligned} \lambda^{\mathsf{p,A}} &= k + \frac{\tau}{n_{\mathsf{A}} - k}, \\ C_{\mathsf{r}}^{\mathsf{p,A}} &= O\bigg((k-1)\nu + k\nu^2 + \frac{\tau\nu}{n_{\mathsf{A}} - k}\bigg), \end{aligned}$$

respectively.

Considering the i-th Class B node, the repair of an erased parity symbol requires downloading $k-\tau-1-i$, $i = 0, \ldots, n_{\mathsf{B}}-k-1$, data symbols. The repair entails $k-\tau-2-i$ additions, and the average normalized repair bandwidth $\lambda^{\mathsf{p,B}}$ and repair complexity $C_{\mathsf{r}}^{\mathsf{p,B}}$ are given as

$$\begin{aligned} \lambda^{\mathsf{p,B}} &= \frac{\sum_{i=0}^{n_{\mathsf{B}}-k-1} k-\tau-1-i}{n_{\mathsf{B}}-k} = \frac{1}{2}\bigg(3k-2\tau-n_{\mathsf{B}}-1\bigg), \\ C_{\mathsf{r}}^{\mathsf{p,B}} &= O\bigg(\frac{\sum_{i=0}^{n_{\mathsf{B}}-k-1} k-\tau-2-i}{n_{\mathsf{B}}-k}\nu\bigg) \\ &= O\bigg(\frac{1}{2}(3k-2\tau-n_{\mathsf{B}}-3)\nu\bigg). \end{aligned}$$

F. Encoding Complexity

The encoding complexity, denoted by C_{e}, is the sum of the encoding complexities of Class A and Class B codes. The generation of each of the $n_{\mathsf{A}}-k$ Class A parity symbols in one row of the code array, $p_{i,j}^{\mathsf{A}}$ in (2), requires k multiplications and $k-1$ additions. Adding data symbols to τ of these parity symbols according to (3) requires an additional τ additions. The encoding complexity of the Class A code is therefore

$$C_{\mathsf{e}}^{\mathsf{A}} = O((n_{\mathsf{A}}-k)(k\nu^2 + (k-1)\nu)) + O(\tau\nu).$$

According to Sections IV and V, the parity symbols in the first Class B parity node are constructed as sums of at most $k-\tau-1$ data symbols, and each parity symbol in the subsequent parity nodes is constructed as a sum of data symbols from a set of size one less. Therefore, the encoding complexity of the Class B code is

$$\begin{aligned} C_{\mathsf{e}}^{\mathsf{B}} &\leq \sum_{i=1}^{n-n_{\mathsf{A}}} O((k-\tau-1-i)\nu) \\ &= O\bigg(\frac{1}{2}(n-n_{\mathsf{A}})(2k-2\tau-3-n+n_{\mathsf{A}})\nu\bigg). \end{aligned} \tag{10}$$

Note that for Construction 1 the upper bound on $C_{\mathsf{e}}^{\mathsf{B}}$ in (10) is tight. Finally, $C_{\mathsf{e}}^{\mathsf{s}} = C_{\mathsf{e}}^{\mathsf{A}} + C_{\mathsf{e}}^{\mathsf{B}}$.

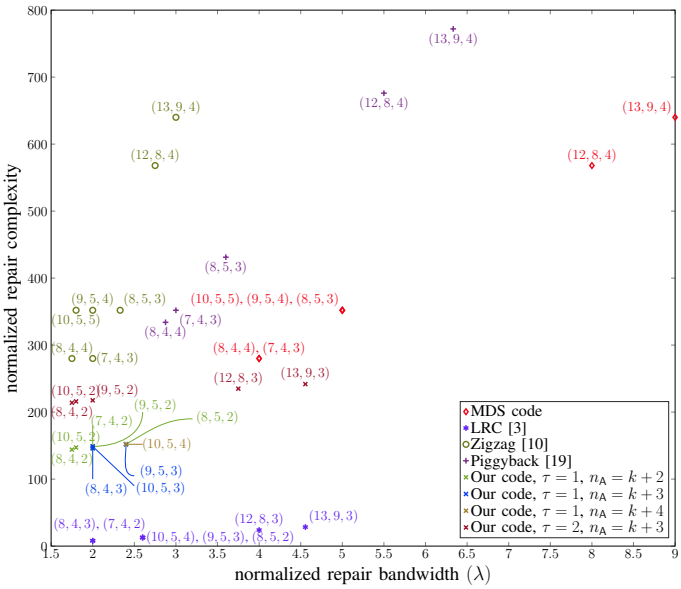

Fig. 5. Comparisons of different (n, k, f) codes with $\nu = 8$.

G. Code Comparison

In this section, we compare the performance of the proposed codes with that of several codes in the literature, namely MDS codes, exact-repairable MDS codes [24], MDR codes [9], Zigzag codes [10], Piggyback codes [19], generalized Piggyback codes [23], EVENODD codes [25], Pyramid codes [2], and LRCs [3]. Throughout this section, we compare the repair bandwidth and the repair complexity of the systematic nodes with respect to other codes, except for exact-repairable MDS and BASIC PM-MBR codes. The reported repair bandwidth and complexity for these codes are for all nodes (both systematic and parity nodes).

Table I provides a summary of the characteristics of the proposed codes as well as different codes proposed in the literature.[4] In the table, column 2 reports the value of β (see (1)) for each code construction. For our code, $\beta = k$, unlike for MDR and Zigzag codes, for which β grows exponentially with k. This implies that our codes require less memory to cache data symbols during repair. On the contrary, EVENODD codes have a lower sub-packetization and repair complexity, but this comes at the cost of having the same repair bandwidth as MDS codes. The Piggyback codes presented in the table and throughout this section are from the piggybacking design 1 in [19], which provides efficient repair for only data nodes. The fault tolerance f, the normalized repair bandwidth λ, the normalized repair complexity, and the encoding complexity, discussed in the previous subsections, are reported in columns 3, 4, 5, and 6, respectively.

In Figs. 5 and 6, we compare our codes with Construction 1 for the Class B codes (i.e., the codes are constructed as shown in Sections III and IV) with other codes in the literature. We remark that the Pyramid codes in Fig. 6 refer to the basic Pyramid codes in [2], while the exact-repairable MDS codes refer to the $(2, n-k, n-1)$ exact-repairable MDS codes from [24, Sec. IV]. The aforementioned notation, unlike our notation in this paper, refers to an $(n, k, n-k)$ code that has $\lambda \leq 2\frac{n-1}{n-k}$, $\beta = n-k$, and repair locality of $n-1$. For generalized Piggyback codes [23], we choose $\beta = k$. Also note that the parameters s, p are chosen according to [23, Eq. 20], i.e., $s = \left\lfloor k\frac{\sqrt{n-k}-1}{\sqrt{n-k}} \right\rfloor$ or $s = \left\lceil k\frac{\sqrt{n-k}-1}{\sqrt{n-k}} \right\rceil$ and $p = k - s$, whichever pair of values gives the lowest repair bandwidth. In case of a tie, the pair that gives the lowest repair complexity was chosen. In particular, the figure plots the normalized repair complexity of (n, k, f) codes over GF(2^8) $(\nu = 8)$ versus their normalized repair bandwidth λ. In the figure, we show the exact repair bandwidth for our proposed codes, while the reported repair complexities and the repair bandwidths of the other codes, except for Piggyback, generalized Piggyback, and exact-repairable MDS codes, are from Table I.[5] For Piggyback, generalized Piggyback, and exact-repairable MDS codes exact values for the repair bandwidth and the repair complexity are calculated directly from the codes. Furthermore, for a fair comparison we assume the parity symbols in the first parity node of all storage codes to be weighted sums. The only exception is the LRCs and the exact-repairable MDS codes, as the code design enforces the parity-check equations to be non-weighted sums. Thus, changing it would alter the maximum erasure correcting capability of the LRC and the repair bandwidth of the exact-repairable MDS code. We also assume that the LRCs and the Pyramid codes have a repair locality of $k/2$. For the generalized Piggyback codes, we assume that the codes have sub-packetization $\beta = k$. For the Piggyback codes, we consider the construction that repairs just the data nodes. Therefore, they have a sub-packetization of 2.

The best codes for a DSS should be the ones that achieve the lowest repair bandwidth and have the lowest repair complexity. As seen in Fig. 5, MDS codes have both high repair complexity and repair bandwidth, but they are optimal in terms of fault tolerance for a given n and k. Zigzag codes achieve the same fault tolerance and high repair complexity as MDS codes, but at the lowest repair bandwidth. At the other end, LRCs yield the lowest repair complexity, but a higher repair bandwidth and worse fault tolerance than Zigzag codes. Piggyback codes, generalized Piggyback codes, and exact-repairable MDS codes have a repair bandwidth between those of Zigzag and MDS codes, but with a higher repair complexity. Strictly speaking, they have a repair complexity higher than MDS codes. For a given storage overhead, our proposed codes have better repair bandwidth than MDS codes, Piggyback codes, generalized Piggyback codes, and exact-repairable MDS codes. In particular, the numerical results in Figs. 5 and 6 show that for different code parameters with $\tau = 1$ and $n_{\mathsf{A}} - k = 2$ our proposed codes yield a reduction of the repair bandwidth in the range of $64\% - 50\%$, $39.13\% - 33.33\%$, $43.04\% - 33.33\%$, and $33.33\% - 30\%$, respectively. Furthermore, our proposed codes yield lower repair complexity as compared to MDS, Piggyback, generalized Piggyback, exact-repairable MDS, and Zigzag codes. Again, the numerical analysis in Figs. 5 and 6

[4]The variables (t, t_r) and r in Table I are defined in [19] and [3], respectively. The definition of ℓ comes directly from r that is defined in [19].

[5]For LRCs the expressions for the repair bandwidth and the repair complexity tabulated in Table I are used when $n-k-r$ is a divisor of k. When $n-k-r$ is not a divisor of k, exact values for the repair bandwidth and the repair complexity are calculated directly from the codes.

TABLE II
COMPARISON OF NORMALIZED REPAIR COMPLEXITY AND BANDWIDTH OF (n, k, f) BASIC PM-MBR CODES [20] AND THE PROPOSED CODES.

Proposed	n_{A}	τ	R	GF(q)	**BASIC PM-MBR**	R^{b}	δ^{b}	$\mathcal{R}_m$	$C_{\mathsf{r}}^{\mathsf{b}}$	C_{r}	λ^{b}	λ
$(9, 5, 3)$	8	1	0.5556	GF(11)	$(8, 5, 3)$	0.4464	7	$\mathcal{R}_{11}$	135	44	1	2.4
$(11, 7, 3)$	10	2	0.6364	GF(11)	$(11, 7, 4)$	0.4454	10	$\mathcal{R}_{11}$	187.5	66.2857	1	3
$(14, 9, 3)$	12	2	0.6428	GF(13)	$(14, 9, 5)$	0.4450	13	$\mathcal{R}_{17}$	384	70.6667	1	3.5556

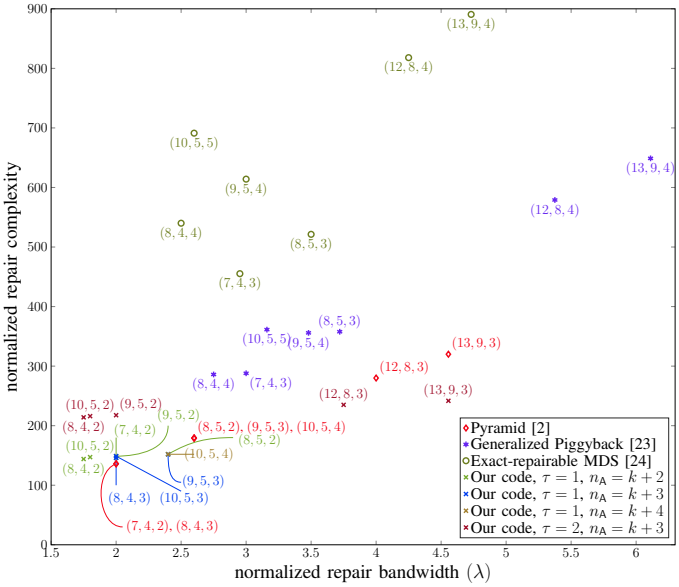

Fig. 6. Comparisons of different (n, k, f) codes with $\nu = 8$.

shows that for different code parameters with $\tau = 1$ and $n_{\mathsf{A}} - k = 2$ our proposed codes yield a reduction of the repair complexity in the range of $58.18\% - 47.86\%$, $70.90\% - 55.23\%$, $59.26\% - 49.30\%$, $78.70\% - 67.93\%$, and $58.18\% - 47.86\%$, respectively. However, the benefits in terms of repair bandwidth and/or repair complexity with respect to MDS codes, Zigzag codes, and codes constructed using the piggybacking framework come at a price of a lower fault tolerance. For fixed (n, k, f) it can be seen that the proposed codes yield a reduction of the repair bandwidth in the range of $7.69\% - 0\%$ compared to LRCs and Pyramid codes, while in some cases, for the latter codes our proposed codes achieve a reduction of the repair complexity in the range of $24.44\% - 15.18\%$.

In Table II, we compare the normalized repair complexity of the proposed codes with Construction 1 for the Class B code and BASIC PM-MBR codes presented in [20]. BASIC PM-MBR codes are constructed from an algebraic ring $\mathcal{R}_m$, where each symbol in the ring is a binary vector of length m. In order to have a fair comparison with our codes, we take the smallest possible field size for our codes and the smallest possible ring size for the codes in [20]. Furthermore, to compare codes with similar storage overhead, we consider BASIC PM-MBR codes with repair locality, denoted by δ^{b}, that leads to the same file size as for our proposed codes (which is equal to k^2) and code length n such that the code rate (which is the inverse of the storage overhead) is as close as possible to that of our proposed codes. The Class A codes of the proposed codes in the table are $(n_{\mathsf{A}}, n_{\mathsf{A}} - f)$ RS codes. The codes are over GF$(n_{\mathsf{A}} + 1)$ if $n_{\mathsf{A}} + 1$ is a prime (or a power of a prime). If $n_{\mathsf{A}} + 1$ is not a prime (or a power of prime), we construct an $(n', n' - f)$ RS code over GF$(n' + 1)$, where $n' + 1$ is the smallest prime (or the smallest power of a prime) with $n' + 1 \geq n_{\mathsf{A}} + 1$, and then we shorten the RS code to obtain an $(n_{\mathsf{A}}, n_{\mathsf{A}} - f)$ Class A code. In the table, the parameters for the proposed codes are given in columns 1 to 5 and those of the codes in [20] in columns 6 to 9. The code rates R and R^{b} of the proposed codes and the BASIC PM-MBR codes[6] are given in columns 4 and 7, respectively. The smallest possible field size for our codes and the smallest possible ring size for the BASIC PM-MBR codes are given in columns 5 and 9, respectively. The normalized repair complexity[7] for BASIC PM-MBR codes, $C_{\mathsf{r}}^{\mathsf{b}}$, is given in column 10, while the normalized repair complexity of the proposed codes, C_{r}, is given in column 11. The normalized repair bandwidth for BASIC PM-MBR codes, λ^{b}, is given in column 12, while the normalized repair bandwidth of the proposed codes, λ, is given in the last column. It can be seen that the proposed codes achieve significantly better repair complexity. However, this comes at the cost of a lower fault tolerance for the codes in rows 2 and 3 (but our codes have significantly higher code rate) and higher repair bandwidth (since BASIC PM-MBR codes are MBR codes, their normalized repair bandwidth is equal to 1). For the $(9, 5, 3)$ code, the same fault tolerance of the $(8, 5, 3)$ code in [20] is achieved, despite the fact that the proposed code has a higher code rate. We remark that FR codes achieve a better (trivial) repair complexity compared to our proposed codes. However, this comes at a cost of $R < 0.5$ and they cannot be constructed for any n, k, and δ, where δ is the repair locality.

In Table III, we compare the normalized repair bandwidth of the proposed codes using Construction 1 and Construction 2 for Class B nodes. In the table, with λ^{C1} and λ^{C2}, we refer to the normalized repair bandwidth for Construction 1 and Construction 2, respectively. For the codes presented, it is seen that the heuristic construction yields an improvement in repair bandwidth in the range of $1\% - 7\%$ with respect to Construction 1.

VII. CONCLUSION

In this paper, we constructed a new class of codes that achieve low repair bandwidth and low repair complexity for a single node failure. The codes are constructed from two smaller codes, Class A and B, where the former focuses on the fault tolerance of the code, and the latter focuses on reducing the repair bandwidth and complexity. It is numerically seen

[6]BASIC PM-MBR codes have code rate $R^{\mathsf{b}} = B/n\delta^{\mathsf{b}}$, where $B = \binom{k+1}{2} + k(\delta^{\mathsf{b}} - k)$ is the file size and $\delta^{\mathsf{b}} \in \{k, k+1, \ldots, n-1\}$.

[7]The normalized repair complexity of BASIC PM-MBR codes is $C_{\mathsf{r}}^{\mathsf{b}} = (3.5\delta^{\mathsf{b}} + 2.5)(m-1)/2$ [20]. Such codes have $\beta = \delta^{\mathsf{b}}$, $\lambda = 1$, and $f = n - k$. The value of m is conditioned on the code length n (see [20, Th. 14]).

TABLE III
IMPROVEMENT IN NORMALIZED REPAIR BANDWIDTH OF THE PROPOSED (n,k) CODES WHEN THE CLASS B NODES ARE HEURISTICALLY CONSTRUCTED.

Code	n_{A}	τ	λ^{C1}	λ^{C2}	**Improvement**
$(7,4)$	6	1	2	1.875	6.25%
$(10,6)$	9	2	2.5	2.4167	3.33%
$(13,8)$	12	3	3	2.9375	2.08%
$(14,8)$	12	3	2.375	2.3125	2.63%
$(16,10)$	15	4	3.5	3.45	1.43%

that our proposed codes achieve better repair complexity than Zigzag codes, MDS codes, Piggyback codes, generalized Piggyback codes, exact-repairable MDS codes, BASIC PM-MBR codes, and are in some cases better than Pyramid codes. They also achieve a better repair bandwidth compared to MDS codes, Piggyback codes, generalized Piggyback codes, exact-repairable MDS codes, and are in some cases better than LRCs and Pyramid codes. A side effect of such a construction is that the number of symbols per node that need to be encoded grows only linearly with the code dimension. This implies that our codes are suitable for memory constrained DSSs as compared to Zigzag and MDR codes, for which the number of symbols per node increases exponentially with the code dimension.

APPENDIX A
PROOF OF THEOREM 1

Consider an arbitrary set of $\tau' \leq \tau$ piggybacked nodes, denoted by $\mathcal{T} = \{j_1, j_2, \ldots, j_{\tau'}\}$, $j_i = j_i' - (n_{\mathsf{A}} - \tau) + 1$, $j_i' \in \{n_{\mathsf{A}} - \tau, \ldots, n_{\mathsf{A}} - 1\}$. Then, the repair of the i-th row, $i = 0, \ldots, k-1$, using the piggybacked nodes in $\mathcal{T}$, would depend upon the knowledge of the data symbols (piggybacks) in the rows $(i+j_1)_k, (i+j_2)_k, \ldots, (i+j_{\tau'})_k$. This is because the knowledge of the piggybacks in these rows allows to obtain the original MDS parity symbols in the i-th row. In the following, we use this observation. We first proceed to prove that if $\tau'(\tau'+n_{\mathsf{A}}-k-\tau) < k$, then $\theta+\tau' \leq n_{\mathsf{A}}-k-\tau+\tau'$ data nodes can be corrected using θ non-modified parity nodes and the τ' piggybacked nodes in $\mathcal{T}$. Using this we will complete the proof of Theorem 1.

Lemma 1: Consider an (n_{A}, k) Class A code with $k+2 \leq n_{\mathsf{A}} < 2k$. The code consists of τ piggybacked nodes and $n_{\mathsf{A}} - k - \tau$ non-modified MDS parity nodes. If $\tau'(\tau' + n_{\mathsf{A}} - k - \tau) < k$, then the code can correct $\theta + \zeta$ data node failures using $\theta \leq n_{\mathsf{A}} - k - \tau$ non-modified parity nodes and the τ' piggybacked parity nodes in the set $\mathcal{T} = \{1, \ldots, \tau'\}$ for $\zeta \leq \tau' \leq \tau$.

Proof: We consider first the case when $\zeta = \tau'$. Then, assume that $\theta + \zeta$ data nodes fail and there exists a sequence $i, (i+1)_k, \ldots, (i-1+\tau')_k$ of τ' data nodes that are available. By construction, the parity symbol $p^{\mathsf{A},\mathsf{p}}_{(j-1)_k, n_{\mathsf{A}}-\tau-1+t}$, $t \in \mathcal{T}$, is (see (3)) given by

$$p^{\mathsf{A},\mathsf{p}}_{(j-1)_k, n_{\mathsf{A}}-\tau-1+t} = p^{\mathsf{A}}_{(j-1)_k, n_{\mathsf{A}}-\tau-1+t} + d_{(j-1+t)_k,(j-1)_k}, \tag{11}$$

where $j = 0, \ldots, k-1$. To recover all data symbols, set $i' = (i+\tau')_k$ and perform the following steps.

1) Obtain the $\zeta = \tau'$ MDS parity symbols $p^{\mathsf{A}}_{(i'-1)_k, n_{\mathsf{A}}-\tau-1+t}$, $t \in \mathcal{T}$ (see (11)) in the $(i'-1)_k$-th row. This is possible because the piggybacks in the $(i'-1)_k$-th node are available.
2) Using the $\theta+\zeta$ MDS parity symbols and $k-\theta-\zeta$ data symbols in the $(i'-1)_k$-th row, recover the missing $\theta+\zeta$ symbols in the $(i'-1)_k$-th row of the failed data nodes.
3) $i' \leftarrow (i'-1)_k$.
4) Repeat Items 1), 2), and 3) $\tau'-1$ times. This ensures that the failed symbols in the τ' rows $i', (i'+1)_k, \ldots, (i'-1+\tau')_k$ are recovered. This implies that the piggyback symbols

$$\begin{aligned} &d_{i',(i'-1)_k}, \ldots, d_{(i'-1+\tau')_k,(i'-1)_k} \text{ in node } (i'-1)_k, \\ &d_{i',(i'-2)_k}, \ldots, d_{(i'-2+\tau')_k,(i'-2)_k} \text{ in node } (i'-2)_k, \\ &\qquad\vdots \\ &d_{i',(i'-\tau')_k} \qquad\qquad\qquad\qquad \text{in node } (i'-\tau')_k, \end{aligned} \tag{12}$$

are recovered. In other words, (12) says that in the $(i'-t)_k$-th node, $\tau'+1-t$ piggybacked symbols are recovered. More specifically, for $t=1$, τ' piggybacked symbols are recovered. Thus,
5) repeat Items 1) and 2), and set
6) $i' \leftarrow (i'-1)_k$. We thus recover the i'-th row and obtain the piggyback symbols in $\mathcal{D}_{\mathcal{R}_{i'}} \backslash \mathcal{D}_{\mathcal{X}_{i'}}$. This increases the number of obtained piggyback symbols by 1 in the next τ' nodes $i', (i'-1)_k, \ldots, (i'+\tau'-1)_k$. In a similar fashion, we now have τ' piggybacked symbols in the i'-th node, and Items 5) and 6) are repeated until all k rows have been recovered. With this recursion, one recovers the $\theta + \tau' = \theta + \zeta$ failed data nodes.

For the case when $\zeta < \tau'$, the aforementioned decoding procedure is still able recover $\theta+\zeta$ data node failures. This is because, in order to repair failed symbols in the $(i'-1)_k$-th row, one needs just $\zeta < \tau'$ MDS parity symbols from $\mathcal{T}' \subseteq \mathcal{T}$. If the $(i'-1)_k$-th node is available, then ζ piggybacks allow to recover the MDS parity symbols (see Item 1)). On the contrary, if the $(i-1)_k$-th node has been erased, then ζ piggybacks are obtained from (12).

Note that the argument above assumes that τ' consecutive data nodes are available. Thus, in order to guarantee that any $\theta+\zeta$ data nodes can be corrected, we consider the worst case scenario for node failures, where we equally spread $\theta+\zeta$ data node failures across the k data nodes. Since

$$\frac{k}{\theta+\zeta} \geq \frac{k}{n_{\mathsf{A}}-k-\tau+\tau'} > \tau',$$

where the last inequality follows by the assumption on τ' stated in the lemma, it follows that the largest gap of non-failed data nodes in the worst case scenario is indeed greater than or equal to τ'. ∎

The above lemma shows that the Class A code can correct up to $n_{\mathsf{A}}-k-\tau+\tau'$ erasures using non-modified parity nodes and τ' modified parity nodes, provided the condition on τ' is satisfied.

To prove that the code can correct $n_{\mathsf{A}}-k-\tau+\tau'$ arbitrary node failures, let us assume that ρ Class A parity nodes and $n_{\mathsf{A}}-k-\tau+\tau'-\rho$ data nodes have failed. More precisely, let $\rho_1 \leq n_{\mathsf{A}}-k-\tau$ non-modified nodes, $\rho_2 \leq \tau'$ piggybacked

Algorithm 1: Class B node construction when k is even

Initialization:
$\boldsymbol{A} = [a_{i,j}]$ as defined in (7)
$n = n_{\mathsf{A}} + n_{\mathsf{B}} - k$ with $n_{\mathsf{B}} < 2k - \tau$ and $n_{\mathsf{A}} < 2k$
for $l \in \{n_{\mathsf{A}}, \ldots, n-1\}$ **do**
  $\rho_l \leftarrow k - \tau - 1 - l + n_{\mathsf{A}}$
  if $l \leq n_{\mathsf{A}} + k/2 - \tau - 2$ **then**
    $\mathcal{P}_l^{\mathsf{B,h}} \leftarrow \mathcal{P}_l^{\mathsf{B}}$
  else if $l > n_{\mathsf{A}} + k/2 - \tau - 2$ and $\rho_l > 1$ **then**
    $\mathcal{P}_l^{\mathsf{B,h}} \leftarrow$ `ConstructNode(`$\boldsymbol{A}, \rho_l$`)`
  else if $l > n_{\mathsf{A}} + k/2 - \tau - 2$ and $\rho_l = 1$ **then**
    $\mathcal{P}_l^{\mathsf{B,h}} \leftarrow$ `ConstructLastNode()`
  end
  $\boldsymbol{A} \leftarrow$ `UpdateReadCost()`
end

nodes in $\mathcal{T}$, and $\rho_3 \geq 0$ remaining piggybacked nodes, where $\rho_1 + \rho_2 + \rho_3 = \rho$, fail. Clearly, it can be seen that there are $n_{\mathsf{A}} - k - \tau - \rho_1$ non-modified parity nodes and a set of $\zeta = \tau' - \rho_2$ modified nodes $\mathcal{T}' \subseteq \mathcal{T}$ available. Also, note that the number of data node failures is ρ_3 less than the number of combined available piggybacked nodes in $\mathcal{T}'$ and available non-modified parity nodes. Thus, using Lemma 1 with $\theta = n_{\mathsf{A}} - k - \tau - \rho_1 - \rho_3$ and $\zeta = \tau' - \rho_2$, it follows that $\theta + \zeta = (n_{\mathsf{A}} - k - \tau - \rho_1 - \rho_3) + (\tau' - \rho_2) = n_{\mathsf{A}} - k - \tau + \tau' - \rho$ data nodes can be repaired. The remaining ρ failed parity nodes can then be repaired using the k^2 data symbols in the k data nodes.

We remark that the decoding procedure in Lemma 1 in essence solves a system of linear equations by eliminating $\tau' k$ variables (piggybacks) in τ' parity nodes. Once the piggybacks are eliminated, the $k(\theta + \tau')$ data symbols are obtained by solving k systems of linear equations. Thus, the decoding procedure is optimal, i.e., it is maximum likelihood decoding.

Consider the quadratic function $\psi(\tau') = \tau'^2 + (n_{\mathsf{A}} - k - \tau)\tau' - k$. According to the proof of Lemma 1 when $\psi(\tau') \geq 0$, the decoding procedure fails as one can construct a failure pattern for the data nodes where the largest separation (in the number of available nodes) between the failed nodes would be strictly smaller than τ'. The largest τ' such that $\psi(\tau') < 0$ can be determined as follows. By simple arithmetic, one can prove that $\psi(\tau')$ is a convex function with a negative minima and with a positive and a negative root. Therefore,

$$0 \leq \tau' < \xi = \frac{\sqrt{(n_{\mathsf{A}} - k - \tau)^2 + 4k} - (n_{\mathsf{A}} - k - \tau)}{2},$$

where ξ is the positive root of $\psi(\tau')$. Furthermore, it may happen that $\tau < \xi$. Therefore, the maximum number of node failures that the code can tolerate is

$$f = \begin{cases} n_{\mathsf{A}} - k - \tau + \left\lfloor \frac{\sqrt{(n_{\mathsf{A}} - k - \tau)^2 + 4k} - (n_{\mathsf{A}} - k - \tau)}{2} \right\rfloor & \text{if } \tau \geq \xi \\ n_{\mathsf{A}} - k & \text{if } \tau < \xi \end{cases}.$$

APPENDIX B
CLASS B PARITY NODE CONSTRUCTION

In this appendix, we give an algorithm that constructs $n_{\mathsf{B}} - k$ Class B parity nodes $\mathcal{P}_{n_{\mathsf{A}}}^{\mathsf{B,h}}, \ldots, \mathcal{P}_{n-1}^{\mathsf{B,h}}$. The algorithm is a heuristic for the construction of Class B parity nodes such that the repair bandwidth of failed nodes is further reduced in comparison with Construction 1 in Section IV.

The nodes $\mathcal{P}_l^{\mathsf{B,h}}$, $l = n_{\mathsf{A}}, \ldots, n-1$, are constructed recursively as shown in Algorithm 1. The k parity symbols in the l-th node are sums of at most ρ_l data symbols $d_{i,j} \in \cup_{j'} \mathcal{D}_{\mathcal{Q}_{j'}}$. The construction of the l-th node for $l \leq n_{\mathsf{A}} + k/2 - \tau - 2$ (see Line 4) is identical to that resulting from Construction 1 in Section IV. The remaining parity nodes $\mathcal{P}_l^{\mathsf{B,h}}$, $l > n_{\mathsf{A}} + k/2 - \tau - 2$, are constructed using the sub-procedures `ConstructNode` $(\boldsymbol{A}, \rho_l)$ and `ConstructLastNode` ( ). After the construction of each parity node, the read costs of the data symbols $d_{i,j}$ are updated by the sub-procedure `UpdateReadCost` ( ). In the following, we describe each of the above-mentioned sub-procedures.

A. `ConstructNode` $(\boldsymbol{A}, \rho_l)$

This sub-procedure allows the construction of the l-th Class B parity node, where each parity symbol in the node is a sum of at most ρ_l data symbols. The algorithm for the sub-procedure is shown in Algorithm 2. Here, the algorithm is divided into two parts. The first part (Line 1 to Line 19) adds at most two data symbols to each of the k parity symbols, while the second part (Line 21 to Line 43) adds at most $\rho_l - 2$ data symbols.

In the first part, each parity symbol $p_{t,l}^{\mathsf{B}}$, $t = 0, \ldots, k-1$, is recursively constructed by adding a symbol $d_{i,j} \in \mathcal{D}_{\mathcal{Q}_j \setminus \cup_{j'} \mathcal{U}_{j'}}$ which has a corresponding read cost $a_{i,j}$ that is the largest among all symbols indexed by $\mathcal{Q}_j \setminus \cup_{j'} \mathcal{U}_{j'}$ (see Line 2 and Line 7). The next symbol added to $p_{t,l}^{\mathsf{B}}$ is $d_{j,i} \in \mathcal{D}_{\mathcal{X}_j}$ if such a symbol exists (Line 4). Otherwise, the symbol added is $d_{j,i_2} \in \mathcal{D}_{\mathcal{X}_j \setminus \cup_{j'} \mathcal{U}_{j'}}$ if such a symbol exists (Line 10). The set $\mathcal{U}_t$ denotes the index set of data symbols from $\mathcal{D}$ that are added to $p_{t,l}^{\mathsf{B}}$. Note that there exist multiple choices for the symbol $d_{i,j}$. A symbol $d_{i,j}$ such that there is a valid $d_{j,i} \in \mathcal{D}_{\mathcal{X}_j}$ is preferred, since it allows a larger reduction of the repair bandwidth.

The second part of the algorithm chooses recursively at most $\rho_l - 2$ data symbols that should participate in the construction of the k parity symbols. The algorithm chooses a symbol $d_{j,i_3} \in \mathcal{D}_{\mathcal{X}_j \setminus \cup_{j'} \mathcal{U}_{j'}} \neq \emptyset$ such that $\mathrm{read}(d_{j,i_3}, p_{t,l}^{\mathsf{B}} + d_{j,i_3}) < a_{j,i_3}$ (see Line 23). In other words, choose data symbols such that their read cost do not increase. It may happen that $\mathcal{D}_{\mathcal{X}_j \setminus \cup_{j'} \mathcal{U}_{j'}} = \emptyset$. If so, select $d_{j_2,i} \in \cup_{j_1} \mathcal{D}_{\mathcal{X}_{j_1} \setminus \cup_{j_1'} \mathcal{U}_{j_1'}}$ such that $d_{i',j_2} \in \mathcal{D}_{\mathcal{U}_t}$ and $a_{j_2,i} > 1$, for some $i' \neq i$, exists, and then add $d_{j_2,i}$ to the parity symbol $p_{t,l}^{\mathsf{B}}$ (see Line 33). If $d_{j_2,i}$ does not exist, then an arbitrary symbol $d_{j_3,i} \in \cup_{j_1} \mathcal{D}_{\mathcal{X}_{j_1} \setminus \cup_{j_1'} \mathcal{U}_{j_1'}}$ is added (see Line 37). This process is then repeated $\rho_l - 2$ times.

B. `ConstructLastNode` ( )

This procedure constructs the l-th Class B parity node that has $\rho_l = 1$. In other words, each parity symbol $p_{t,l}^{\mathsf{B}}$ is a data

symbol $d_{i,j} \in \cup_{j'} \mathcal{D}_{\mathcal{Q}_{j'}}$. The procedure works as follows. First, initialize $\mathcal{U}_t$ to be the empty set for $t = 0, \ldots, k-1$. Then, for $t = 0, \ldots, k-1$, assign first the data symbol $d_{i,j}$ to the parity symbol $p^{\mathsf{B}}_{t,l}$, where $d_{i,j} \in \mathcal{D}_{\Psi(\boldsymbol{A}_{\cup_{j'} \mathcal{Q}_{j'} \setminus \cup_{t'} \mathcal{U}_{t'}})}$, and then subsequently add (i,j) to $\mathcal{U}_t$. Note that for each iteration there may exist several choices for $d_{i,j}$, in which case we can pick one of these randomly.

C. `UpdateReadCost ( )`

After the construction of the l-th node, we update the read costs of all data symbols $d_{i,j} \in \cup_{j'} \mathcal{D}_{\mathcal{Q}_{j'}}$. These updated values are used during the construction of the $(l+1)$-th node. The read cost updates for the parity symbol $p^{\mathsf{B}}_{t,l}$, $t = 0, \ldots, k-1$, are

$$a_{i,j} = \mathrm{read}(d_{i,j}, p^{\mathsf{B}}_{t,l}), \ \ \forall d_{i,j} \in \mathcal{D}_{\mathcal{U}_t}.$$

REFERENCES

[1] J. Huang, X. Liang, X. Qin, P. Xie, and C. Xie, "Scale-RS: An efficient scaling scheme for RS-coded storage clusters," *IEEE Trans. Parallel and Distributed Systems*, vol. 26, no. 6, pp. 1704–1717, Jun. 2015.

[2] C. Huang, M. Chen, and J. Li, "Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems," in *Proc. IEEE Int. Symp. Network Comput. and Appl. (NCA)*, Cambridge, MA, Jul. 2007.

[3] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure coding in Windows Azure storage," in *Proc. USENIX Annual Technical Conf.*, Boston, MA, Jun. 2012.

[4] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "XORing elephants: Novel erasure codes for big data," in *Proc. 39th Very Large Data Bases Endowment (VLDB)*, Trento, Italy, Aug. 2013.

[5] D. S. Papailiopoulos and A. G. Dimakis, "Locally repairable codes," *IEEE Trans. Inf. Theory*, vol. 60, no. 10, pp. 5843–5855, Oct. 2014.

[6] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Trans. Inf. Theory*, vol. 56, no. 9, pp. 4539–4551, Sep. 2010.

[7] K. V. Rashmi, N. B. Shah, and P. V. Kumar, "Optimal exact-regenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction," *IEEE Trans. Inf. Theory*, vol. 57, no. 8, pp. 5227–5239, Aug. 2011.

[8] S. El Rouayheb and K. Ramchandran, "Fractional repetition codes for repair in distributed storage systems," in *Proc. 48th Annual Allerton Conf. Commun., Control, and Comput.*, Monticello, IL, Sep./Oct. 2010.

[9] Y. Wang, X. Yin, and X. Wang, "MDR codes: A new class of RAID-6 codes with optimal rebuilding and encoding," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 5, pp. 1008–1018, May 2014.

[10] I. Tamo, Z. Wang, and J. Bruck, "Zigzag codes: MDS array codes with optimal rebuilding," *IEEE Trans. Inf. Theory*, vol. 59, no. 3, pp. 1597–1616, Mar. 2013.

[11] V. R. Cadambe, C. Huang, J. Li, and S. Mehrotra, "Polynomial length MDS codes with optimal repair in distributed storage," in *Proc. 45th Asilomar Conf. Signals, Syst. and Comput. (ASILOMAR)*, Pacific Grove, CA, Nov. 2011.

[12] G. K. Agarwal, B. Sasidharan, and P. V. Kumar, "An alternate construction of an access-optimal regenerating code with optimal sub-packetization level," in *Proc. 21st Nat. Conf. Commun. (NCC)*, Mumbai, India, Feb. 2015.

[13] J. Li, X. Tang, and U. Parampalli, "A framework of constructions of minimal storage regenerating codes with the optimal access/update property," *IEEE Trans. Inf. Theory*, vol. 61, no. 4, pp. 1920–1932, Apr. 2015.

[14] Z. Wang, I. Tamo, and J. Bruck, "Explicit minimum storage regenerating codes," *IEEE Trans. Inf. Theory*, vol. 62, no. 8, pp. 4466–4480, Aug. 2016.

[15] B. Sasidharan, M. Vajha, and P. V. Kumar, "An explicit, coupled-layer construction of a high-rate MSR code with low sub-packetization level, small field size and all-node repair," Sep. 2016, arXiv: 1607.07335v3. [Online]. Available: https://arxiv.org/abs/1607.07335

[16] M. Ye and A. Barg, "Explicit constructions of optimal-access MDS codes with nearly optimal sub-packetization," *IEEE Trans. Inf. Theory*, vol. 63, no. 10, pp. 6307–6317, Oct. 2017.

[17] J. Li, X. Tang, and C. Tian, "A generic transformation for optimal repair bandwidth and rebuilding access in MDS codes," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Aachen, Germany, Jun. 2017.

[18] N. Raviv, N. Silberstein, and T. Etzion, "Constructions of high-rate minimum storage regenerating codes over small fields," *IEEE Trans. Inf. Theory*, vol. 63, no. 4, pp. 2015–2038, Apr. 2017.

[19] K. V. Rashmi, N. B. Shah, and K. Ramchandran, "A piggybacking design framework for read-and download-efficient distributed storage codes," *IEEE Trans. Inf. Theory*, vol. 63, no. 9, pp. 5802–5820, Sep. 2017.

[20] H. Hou, K. W. Shum, M. Chen, and H. Li, "BASIC codes: Low-complexity regenerating codes for distributed storage systems," *IEEE Trans. Inf. Theory*, vol. 62, no. 6, pp. 3053–3069, Jun. 2016.

[21] H. Hou, P. P. C. Lee, Y. S. Han, and Y. Hu, "Triple-fault-tolerant binary MDS array codes with asymptotically optimal repair," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Aachen, Germany, Jun. 2017.

[22] S. Kumar, A. Graell i Amat, I. Andriyanova, and F. Brännström, "A family of erasure correcting codes with low repair bandwidth and low repair complexity," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, San Diego, CA, Dec. 2015.

[23] S. Yuan and Q. Huang, "Generalized piggybacking codes for distributed storage systems," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Washington, DC, Dec. 2016.

[24] A. S. Rawat, I. Tamo, V. Guruswami, and K. Efremenko, "MDS code constructions with small sub-packetization and near-optimal repair bandwidth," *IEEE Trans. Inf. Theory*, to appear.

[25] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures," *IEEE Trans. Comput.*, vol. 44, no. 2, pp. 192–202, Feb. 1995.

[26] A. Karatsuba and Y. Ofman, "Multiplication of multidigit numbers on automata," *Sov. Phys. Doklady*, vol. 7, no. 7, pp. 595–596, Jan. 1963.

[27] A. Weimerskirch and C. Paar, "Generalizations of the Karatsuba algorithm for efficient implementations," Jul. 2006. [Online]. Available: https://eprint.iacr.org/2006/224.pdf

[28] J. M. Pollard, "The fast Fourier transform in a finite field," *Math. Comput.*, vol. 25, no. 114, pp. 365–374, Apr. 1971.

[29] R. Crandall and C. Pomerance, *Prime Numbers: A Computational Perspective*. Springer, 2001.

[30] S. Gao and T. Mateer, "Additive fast Fourier transforms over finite fields," *IEEE Trans. Inf. Theory*, vol. 56, no. 12, pp. 6265–6272, Dec. 2010.

Algorithm 2: `ConstructNode` $(\boldsymbol{A}, \rho_l)$

Initialization:

$t, j \leftarrow 0$

$\mathcal{U}_{t_1} \leftarrow \emptyset, \quad t_1 = 0, \ldots, k-1$

while $\min_{\forall t_1} |\mathcal{U}_{t_1}| < 2$ **do**

  Select $d_{i,j} \in \mathcal{D}_{\Psi(\boldsymbol{A}_{\mathcal{Q}_j \setminus \cup_{j'} \mathcal{U}_{j'}})}$ s.t. $\exists\, d_{j,i} \in \mathcal{D}_{\mathcal{X}_j}, a_{j,i} > 1$

  if $d_{i,j}$ *exists* **then**

    $p^{\mathsf{B}}_{t,l} \leftarrow d_{i,j} + d_{j,i}$

    $\mathcal{U}_t \leftarrow \mathcal{U}_t \cup \{(i,j),(j,i)\}$; $t \leftarrow t+1$

  else

    Select $d_{i_1,j} \in \mathcal{D}_{\Psi(\boldsymbol{A}_{\mathcal{Q}_j \setminus \cup_{j'} \mathcal{U}_{j'}})}$

    if $d_{i_1,j}$ *exists* **then**

      if $\exists\, d_{j,i_2} \in \mathcal{D}_{\mathcal{X}_j \setminus \cup_{j'} \mathcal{U}_{j'}}$ s.t. $\mathrm{read}(d_{j,i_2}, p^{\mathsf{B}}_{t,l} + d_{j,i_2}) < a_{j,i_2}$ and $a_{j,i_2} > 1$ **then**

        $p^{\mathsf{B}}_{t,l} \leftarrow d_{i_1,j} + d_{j,i_2}$

        $\mathcal{U}_t \leftarrow \mathcal{U}_t \cup \{(i_1,j),(j,i_2)\}$; $t \leftarrow t+1$

      else

        $p^{\mathsf{B}}_{t,l} \leftarrow d_{i_1,j}$

        $\mathcal{U}_t \leftarrow \mathcal{U}_t \cup \{(i_1,j),(-1,-1)\}$; $t \leftarrow t+1$

      end

    end

  end

  $j \leftarrow (j+1)_k$

end

$t, j \leftarrow 0$

while $\min_{\forall t_1} |\mathcal{U}_{t_1}| \leq \rho_l$ **do**

  if $\mathcal{D}_{\mathcal{X}_j \setminus \cup_{j_1} \mathcal{U}_{j_1}} \neq \emptyset$ **then**

    if $\exists\, d_{j,i_3} \in \mathcal{D}_{\mathcal{X}_j \setminus \cup_{j'} \mathcal{U}_{j'}}$ s.t. $\mathrm{read}(d_{j,i_3}, p^{\mathsf{B}}_{t,l} + d_{j,i_3}) < a_{j,i_3}$ and $a_{j,i_3} > 1$ **then**

      $p^{\mathsf{B}}_{t,l} \leftarrow p^{\mathsf{B}}_{t,l} + d_{j,i_3}$

      $\mathcal{U}_t \leftarrow \mathcal{U}_t \cup \{(j,i_3)\}$; $t \leftarrow t+1$

    else if $l > n_{\mathsf{A}}$ **then**

      $\mathcal{U}_t \leftarrow \mathcal{U}_t \cup (-1,-1)\}$; $t \leftarrow t+1$

    end

  else

    if $|\mathcal{U}_t| \leq \rho_l - 1$ **then**

      Select $d_{j_2,i} \in \cup_{j_1} \mathcal{D}_{\mathcal{X}_{j_1} \setminus \cup_{j_1'} \mathcal{U}_{j_1'}}$ s.t. $d_{i',j_2} \in \mathcal{D}_{\mathcal{U}_t}$ and $a_{j_2,i} > 1$ for some $i' \neq i$

      if $d_{j_2,i}$ *exists* **then**

        $p^{\mathsf{B}}_{t,l} \leftarrow p^{\mathsf{B}}_{t,l} + d_{j_2,i}$

        $\mathcal{U}_t \leftarrow \mathcal{U}_t \cup \{(j_2,i)\}$; $t \leftarrow t+1$

      else

        Select $d_{j_3,i} \in \cup_{j_1} \mathcal{D}_{\mathcal{X}_{j_1} \setminus \cup_{j_1'} \mathcal{U}_{j_1'}}$

        $p^{\mathsf{B}}_{t,l} \leftarrow p^{\mathsf{B}}_{t,l} + d_{j_3,i}$

        $\mathcal{U}_t \leftarrow \mathcal{U}_t \cup \{(j_3,i)\}$; $t \leftarrow t+1$;

      end

    end

  end

  $j \leftarrow (j+1)_k$

end

return $\{p^{\mathsf{B}}_{0,l}, \ldots, p^{\mathsf{B}}_{k-1,l}\}$